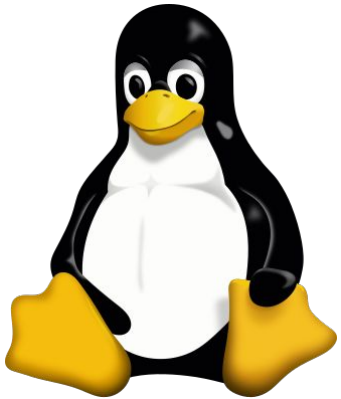
DIR-V GRAND CHALLENGE

VEGA Processor Tutorial

Building Linux for VEGA & Driver access

Centre for Development of Advanced Computing(C-DAC)

Nov 03, 2025



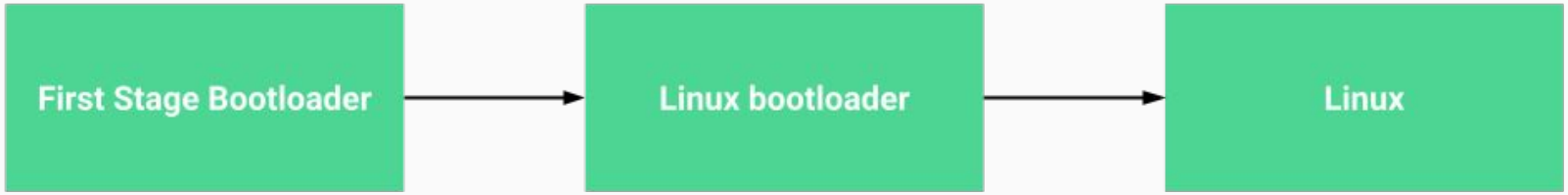
Building Linux for VEGA & Driver access

Participate!

During the lectures ....

- Don’t hesitate to ask questions. Other people in the audience may have similar questions too.
- This helps the trainer to detect any explanation that wasn’t clear or detailed enough.
- Don’t hesitate to share your experience, for example to compare Linux with other operating systems used in your company.
- Your point of view is most valuable, because it can be similar to your colleagues’ and different from the trainer’s.
- Your participation can make out session more interactive and make the topics easier to learn.

Linux boot flow



Bootloader

Available bootloaders :

- RISC-V Proxy Kernel and Boot Loader (riscv-pk)
- OpenSBI
- U-Boot

Here we are using riscv-pk

RISC-V Proxy Kernel and Boot Loader

The RISC-V Proxy Kernel, pk, is a lightweight application execution environment that can host statically-linked RISC-V ELF binaries. It is designed to support tethered RISC-V implementations with limited I/O capability and thus handles I/O-related system calls by proxying them to a host computer.

bbl is expected to have been loaded from first stage boot loader, with the entry point running in machine mode. It is passed a device tree from the prior boot loader stage, and performs the following steps:

- One hart is selected to be the main hart. The other harts are put to sleep until bbl is ready to transfer control to Linux, at which point they will all be woken up and enter Linux around the same time.
- The device tree that was passed in from the previous stage is read and filtered.
- All the other harts are woken up so they can set up their PMPs, trap handlers and enter supervisor mode.

- The mhartid CSR is read so Linux can be passed a unique per-hart identifier.
- Machine mode trap handlers, including a machine mode stack, is set up. bbl's machine mode code needs to handle both unimplemented instructions and machine-mode interrupts.
- The processor executes a mret to jump from machine mode to supervisor mode.
- bbl jumps to the start of its payload, which in this case is Linux.

Compiling Linux for TRISUL32

Buildroot

Buildroot is a set of Makefiles and patches that simplifies and automates the process of building a complete and bootable Linux environment for an embedded system, while using cross-compilation to allow building for multiple target platforms on a single Linux-based development system. Buildroot can automatically build the required cross-compilation toolchain, create a root file system, compile a Linux kernel image, and generate a boot loader for the targeted embedded system, or it can perform any independent combination of these steps. For example, an already installed cross-compilation toolchain can be used independently, while Buildroot only creates the root file system.

Buildroot for TRISUL32 : https://gitlab.com/dirv-vega-gc2025/trisul32_buildroot

Compiling Linux for TRISUL32

For beginners, a simple build script build.sh is provided in buildroot directory.This will build Linux with predefined options.

```
./build.sh
```

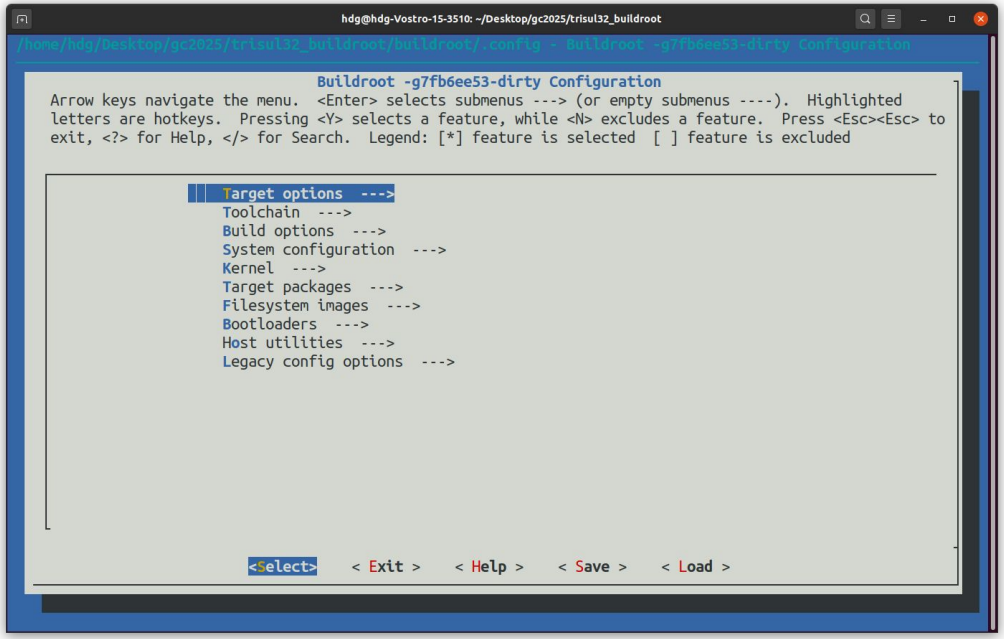
For people with little experience on Linux and buildroot, can try building Linux from scratch.

```
cd buildroot
```

Compiling Linux for TRISUL32

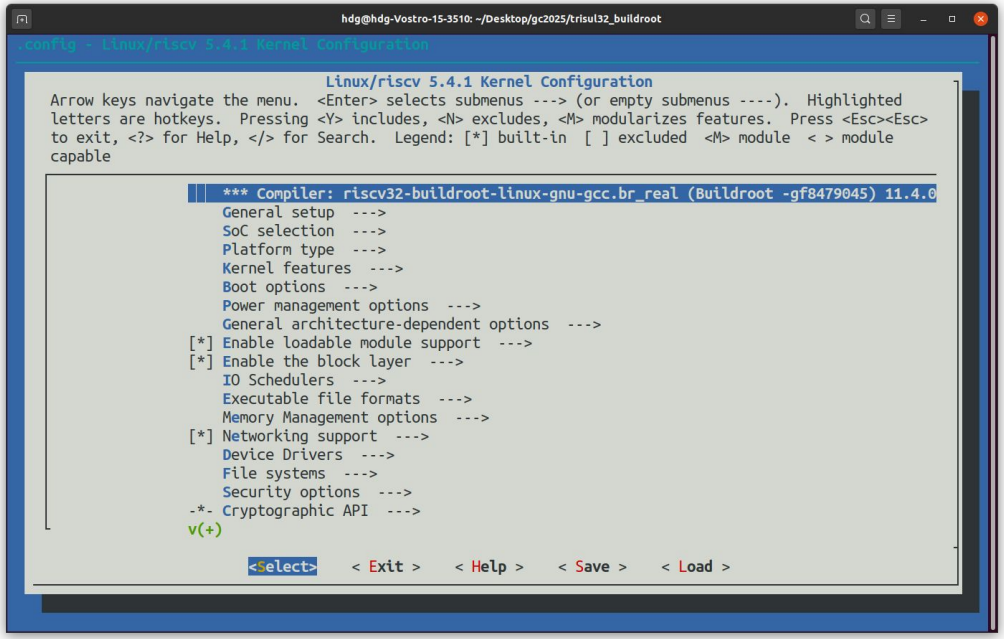
Buildroot menuconfig:

```
make menuconfig
```



Linux menuconfig:

```
make linux-menuconfig
```



Compiling Linux for TRISUL32

After selecting necessary packages, lets build Linux

```
make -j8
```

If build is successful, output/images/ directory will contains final image, vmlinux.
Copy vmlinux to ../riscv-pk/build/.Now you could build Linux binary with Linux bootloader by :

```
../configure --host=riscv32-buildroot-linux-gnu --with-payload=./vmlinux
--with-arch=rv32ima --with-abi=ilp32
```

```
make
```

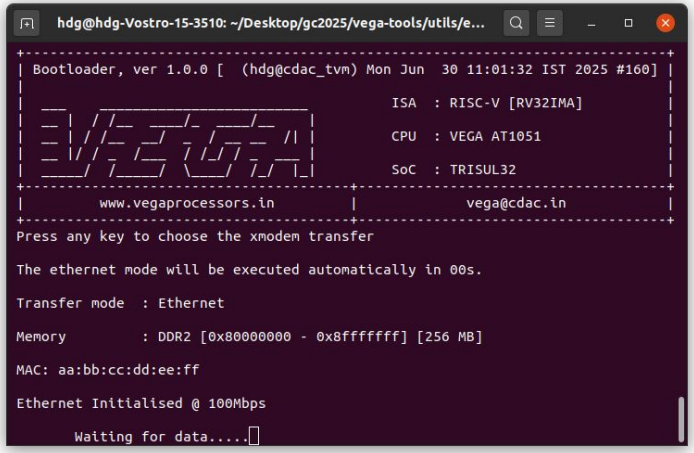
```
riscv32-buildroot-linux-gnu-objcopy --gap-fill 0 -I elf32-littleriscv -O binary
--set-section-flags .bss=alloc,load,contents bbl bbl.bin
```

Sending Linux for TRISUL32

Copy the generated bbl.bin to vega-tools/utils/eth_transfer/ . Now Reset board and open minicom:

```
reset_trisul32.sh
minicom trisul32
```

Wait till board ready for ethernet transfer.



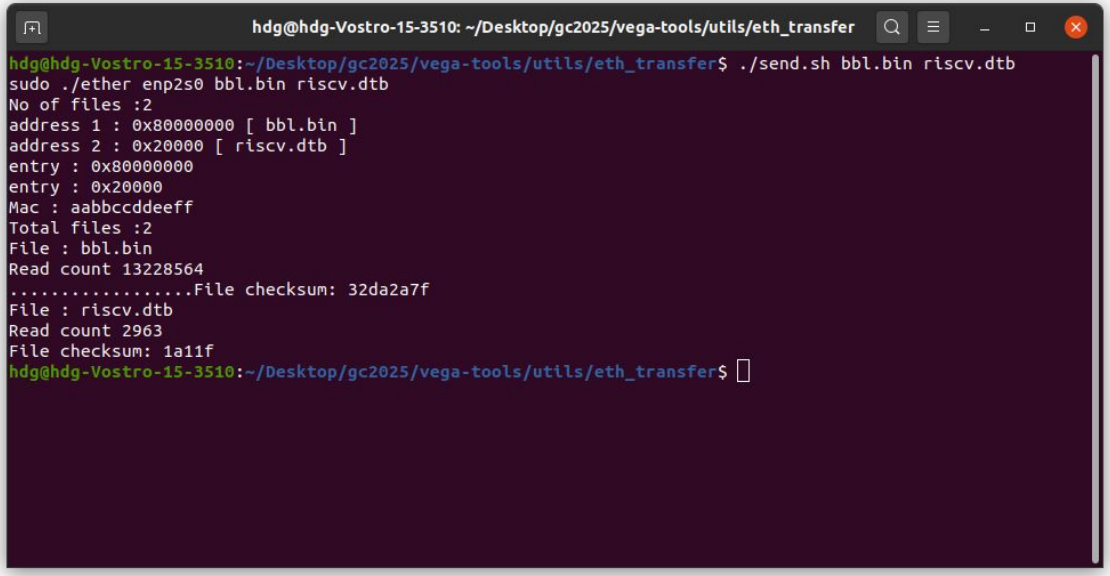
Edit the environment script in vega-tools/utils/eth_transfer/ like below for transfer linux & DTS

```
#Number of files to send
files=2

#Addresses of corresponding files
address1=0x80000000
address2=0x20000

#Entrypoint of program to jump after download
entrypoint=0x80000000

#Device tree binary address
dtbaddress=0x20000

#MAC address of board to set
mac=aa:bb:cc:dd:ee:ff
```

Sending Linux for TRISUL32

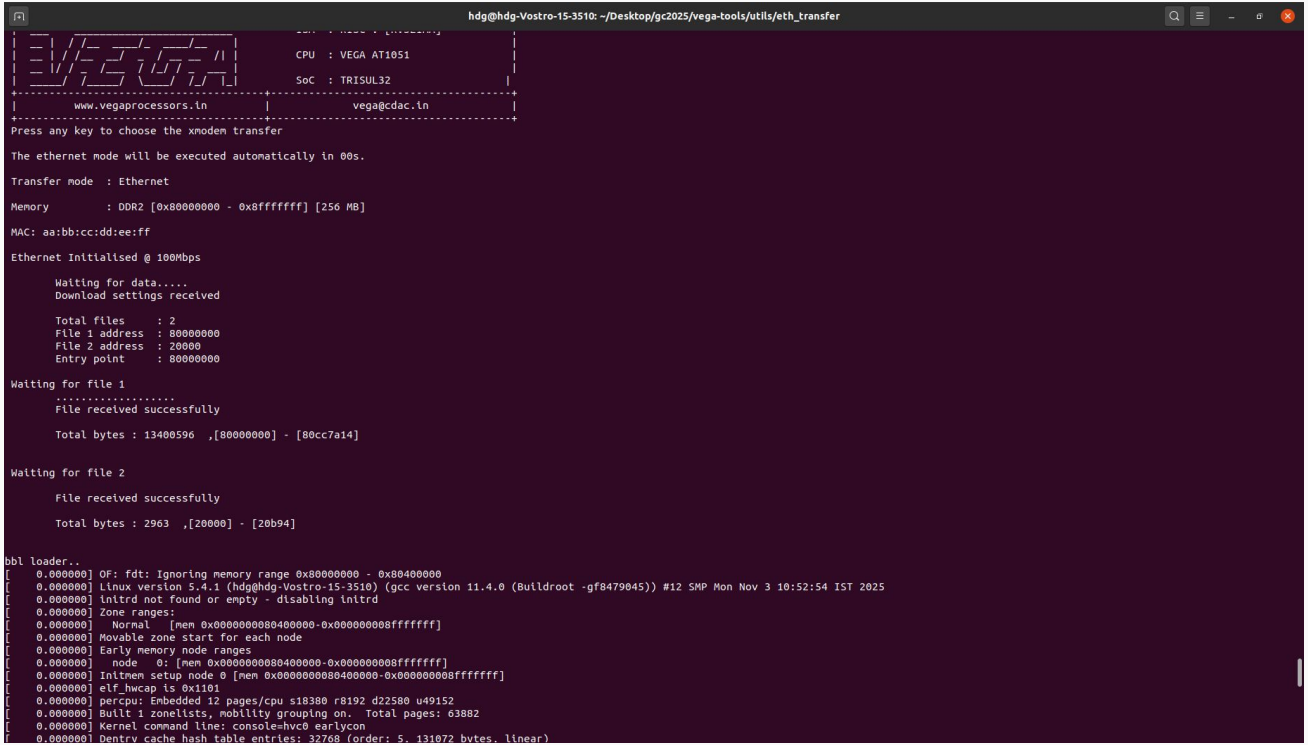
send.sh script in vega-tools/utils/eth_transfer/ will transfer bbl.bin to board.

```
./send.sh bbl.bin riscv.dtb
```



Compiling Linux for TRISUL32

After transfer completed, Linux will start booting.



Peripherals discussed today

- Interrupt Controller *(PLIC)*
- Ethernet *(10/100)*

PLIC

Linux Driver Location :

```
Device Drivers  --->
↓
[*] CDAC Simple Interrupt Controller
```

Linux menuconfig snap shot :

```
        IRQ chip support  ----
    [ ] RISC-V Interrupt Controller
    [ ] SiFive Platform-Level Interrupt Controller
    [ ] CDAC Platform-Level Interrupt Controller
    [*] CDAC Simple Interrupt Controller
```

PLIC device tree node

```
plic0: interrupt-controller@20010000 {
     #interrupt-cells = <1>;
     compatible = "cdac,plic-1.0.0";
     reg = <0x0 0x20010000 0x0 0x4000000>;
     riscv,ndev = <32>;
     interrupt-controller;
     interrupts-extended = <&CPU0_intc 9>;
     };
```

Linux Driver Log :

```
[    0.000000] plic: mapped 32 interrupts to 1 (out of 1) handlers.
```

The /proc interface :

```
# cat /proc/interrupts
           CPU0
  1:       0  CDAC PLIC   6  10000600.spi
  3:       0  CDAC PLIC   8  10000800.i2c
  5:       0  CDAC PLIC   9  10000900.i2c
```

Ethernet

Linux Driver Location :

```
Device Drivers  --->
↓
[*] Network device support  --->
↓
[*]   Ethernet driver support  --->
↓
<*>   CDAC ethernet ERMAC (10/100) on C-DAC devices
```

Linux menuconfig snap shot :

```
        [ ]   Broadcom devices
        [ ]   Cadence devices
        [ ]   Cortina Gemini devices
        < >   CDAC ethernet MAC on CDAC devices
        <*>   CDAC ethernet ERMAC (10/100) on C-DAC devices
        < >   Dave ethernet support (DNET)
        [ ]   EZchip devices
```

Ethernet device tree node

```
ethernet@20030000 {
        device_type = "network";
        compatible = "cdac,cdac_mac";
        interrupt-parent = <&plic0>;
        interrupts = <4>;
        reg = <0x0 0x20030000 0x0 0x100>;
};
```

Linux Driver Log :

```
[   22.224000] CDAC ERMAC driver, version 1.00
[   22.388000] libphy: cdac_mii_bus: probed
[   22.388000] cdac_mac 20030000.ethernet eth0: using MII interface
[   22.400000] Generic PHY 20030000.ethernet-ffffffff:01: attached PHY driver [Generic PHY]
(mii_bus:phy_addr=20030000.ethernet-ffffff)
[   22.412000] cdac_mac 20030000.ethernet eth0: CDAC mac at 0x20030000 irq 1
```

List network interfaces using ip command :

```
# ip addr show
1: lo: <LOOPBACK> mtu 65536 qdisc noop qlen 1000
     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_0
     link/ether aa:bb:cc:dd:ee:ff brd ff:ff:ff:ff:ff:ff
     inet 192.168.7.2/24 brd 192.168.7.255 scope global eth0
     valid_lft forever preferred_lft forever
```

Obtain IP using udhcpc command

```
# udhcpc
udhcpc: started, v1.31.1
[   42.196000] Initializing ethernet....
udhcpc: sending discover
[   43.976000] cdac_mac 20030000.ethernet eth0: link up (100/Half)
udhcpc: sending discover
udhcpc: sending select for 10.176.19.239
udhcpc: lease of 10.176.19.239 obtained, lease time 122
deleting routers
adding dns 10.176.0.11
```

To get information about network interface, eth0

```
# ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
      link/ether 00:a0:22:01:01:8f brd ff:ff:ff:ff:ff:ff
      inet 10.176.19.239/23 brd 10.176.19.255 scope global eth0
      valid_lft forever preferred_lft forever
```

You could transfer program to board via scp.

```
scp hello root@10.176.19.239:/root/
```

Password for root user is root.

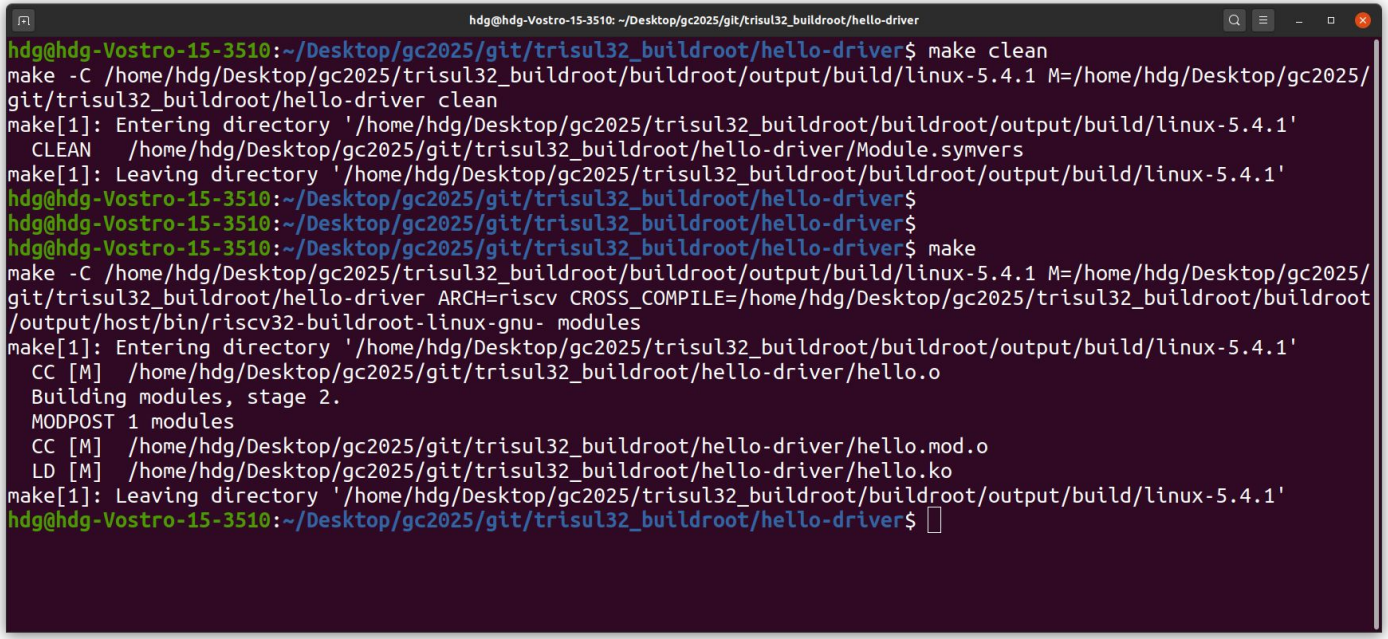
Hello Driver

A sample Hello driver code is provided in trisul32_buildroot folder, Edit the Makefile to compile the kernel module

```
KDIR ?= /<path to linux source>/buildroot/output/build/linux-5.4.1
PWD  := $(shell pwd)

# Default target
obj-m += hello.o

all:
     $(MAKE) -C $(KDIR) M=$(PWD) ARCH=riscv CROSS_COMPILE=/<path to
toolchain>/buildroot/output/host/bin/riscv32-buildroot-linux-gnu- modules

clean:
     $(MAKE) -C $(KDIR) M=$(PWD) clean
```

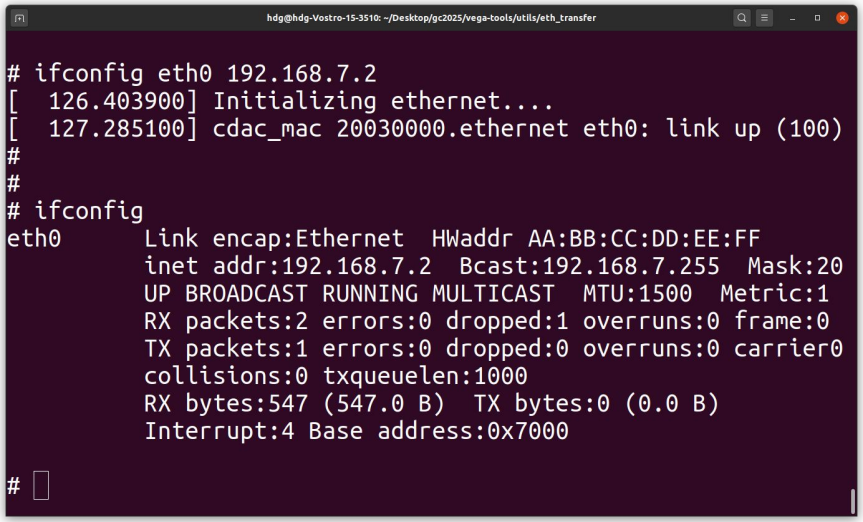
Hello Driver

A sample Hello driver code is provided in trisul32_buildroot folder, Edit the Makefile to compile the kernel module

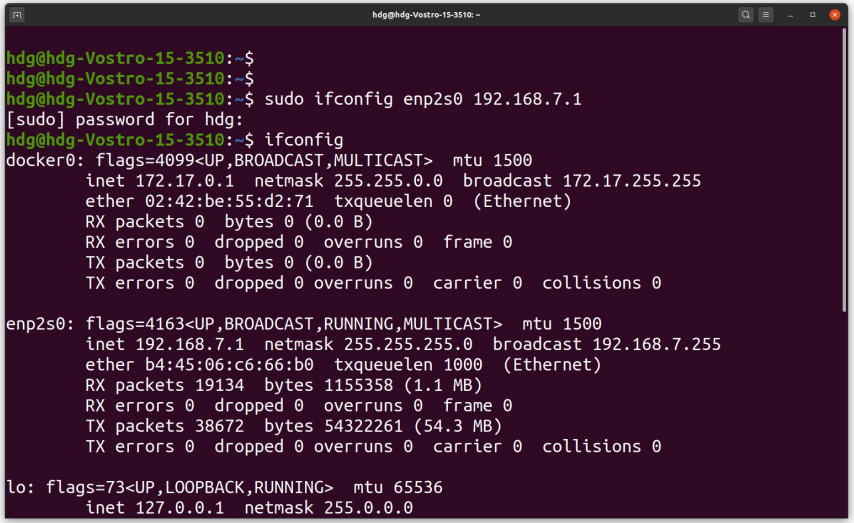


Hello Driver

Copy hello.ko file to board. For that set a local IP for both board and PC



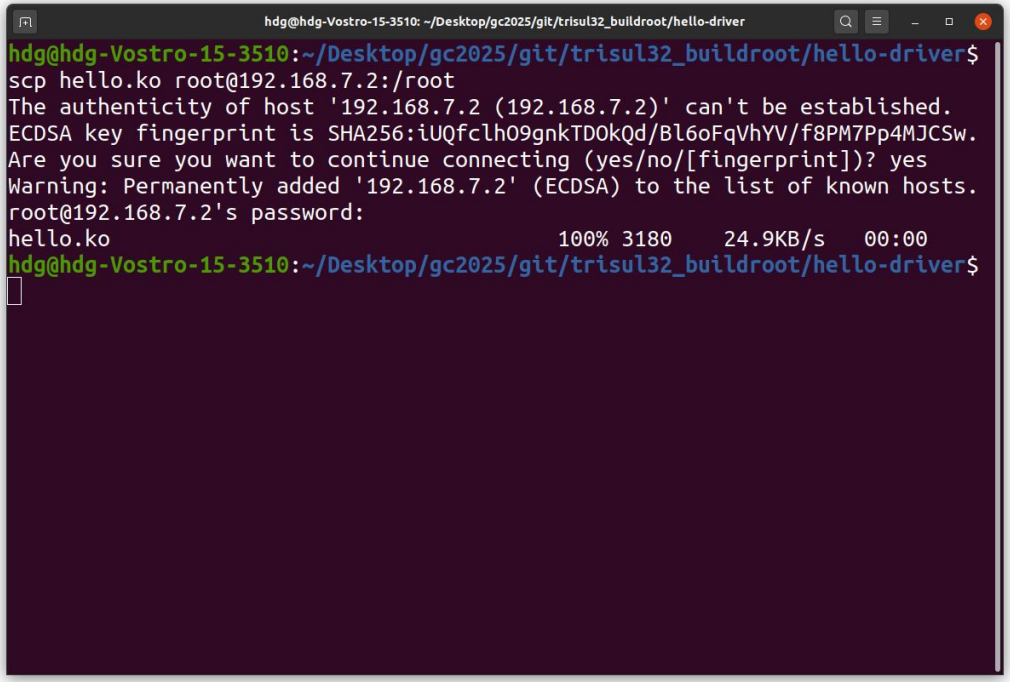
Board



PC

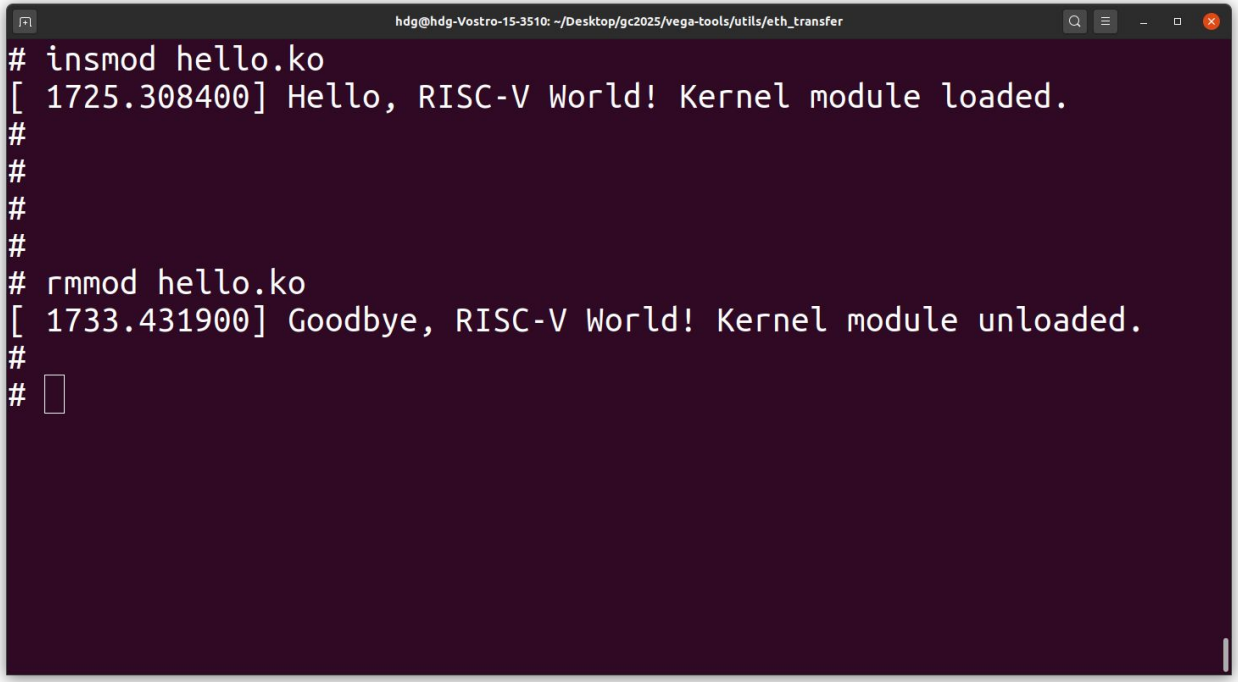
Hello Driver

Now send hello.ko using the scp command from PC.



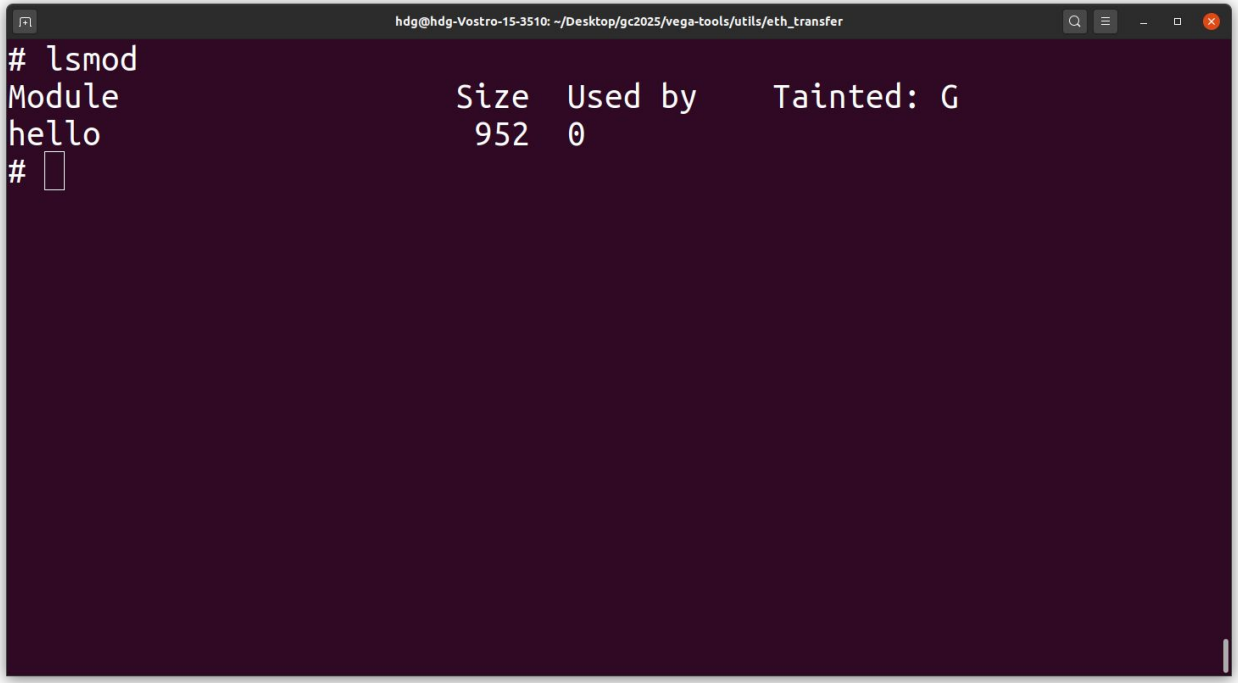
Hello Driver

Insert the kernel module using **insmod** command and **rmmod** for remove the module



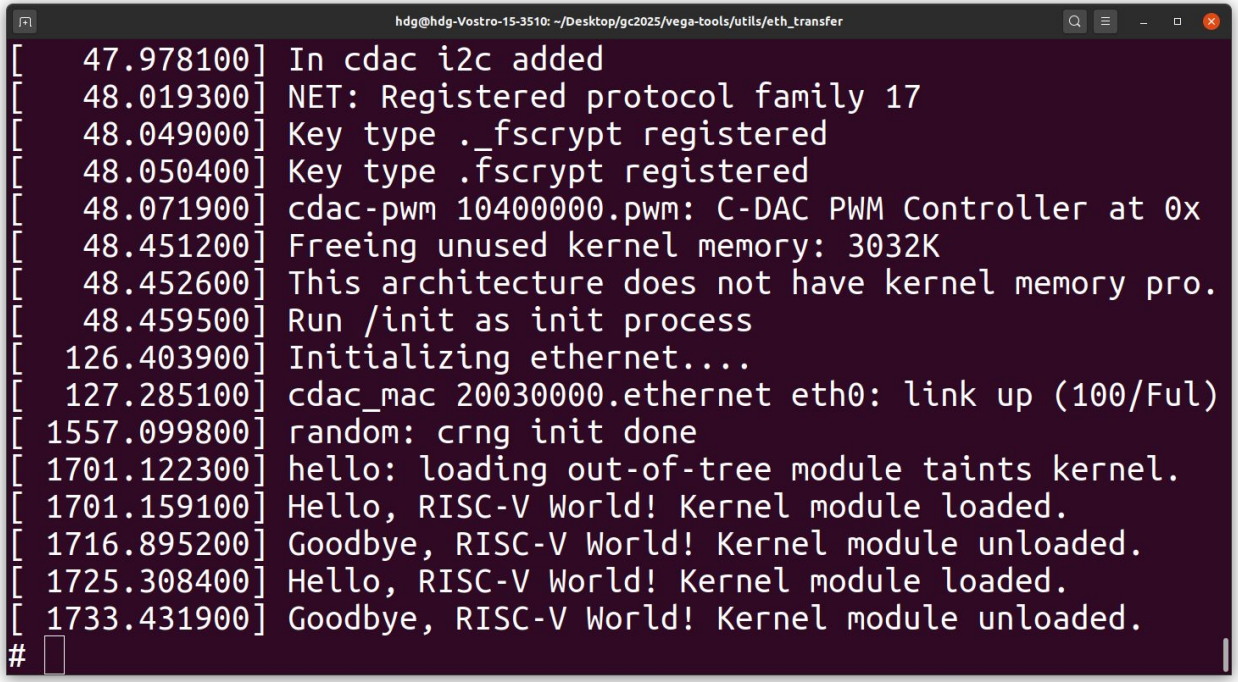
Hello Driver

List the available modules in Linux kernel by executing **lsmod** command



Hello Driver

Also check the **dmesg** for checking the kernel debug messages



Thanks!

Any questions?

You can find us at:

 youtube.com/vegaprocessors

 @vegaprocessor

 www.vegaprocessors.in

 vega@cdac.in